FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Intel Paragon XP/S - Architecture, Software Environment, and Performance

Rudolf Berrendorf, Heribert C. Burg, Ulrich Detert
Rüdiger Esser, Michael Gerndt, Renate Knecht

KFA-ZAM-IB-9409

Mai 1994
(Stand 16.05.94)

Intel Paragon XP/S - Architecture, Software Environment, and Performance

Rudolf Berrendorf, Heribert C. Burg, Ulrich Detert,
Rüdiger Esser, Michael Gerndt, Renate Knecht

Central Institute for Applied Mathematics
Research Centre Jülich (KFA), D-52425 Jülich, Germany
e-mail: r.esser@kfa-juelich.de
Mai 16, 1994

Abstract. The paper describes the hardware and software components of the Intel Paragon XP/S system, a distributed-memory scalable multicomputer. The Paragon processing nodes, which are based on the Intel i860 XP RISC processor, are connected by a two-dimensional mesh with high bandwidth. The paper first gives an overview of the Paragon system architecture, the node architecture, the interconnection network, I/O interfaces, and peripherals. The second part outlines the Paragon OSF/1 operating system and the program development environment including programming models, compilers, application libraries, and tools for parallelization, debugging, and performance analysis. The third part includes performance measurements characterizing the interconnection network, input/output, and numerical performance.

1 Introduction

The Paragon, which was first delivered in Sep tember 1992, is a product of Intel Corporation's Supercomputer Systems Division (Intel SSD). As its predecessors, the prototypical Touchstone Delta system and the iPSC/860, the Paragon is a scalable distributed multicomputer. Its nodes are also based on Intel's i860 RISC processor and the primarily supported programming model is message-passing. The most significant differences between the Paragon and the iPSC/860 with its hypercube topology are the new fast rectangular interconnection network and the new OSF/1 based operating system.

This report intends to give a tutorial survey of the Paragon hardware and software architecture. In addition to providing an update of an earlier version [**?**], it includes an assessment of the Paragon's performance characteristics w.r.t. the interconnection network, input/output, and numerical performance. At the time of writing (May 1994), more than 50 Paragon systems have been delivered to customers including a very large system with 1984 nodes. The hardware of the Paragon has reached a certain maturity, whereas the software is still under development. A large part of the functionality announced for the Paragon has been available since fall 1993 when Release 1.1 of the Paragon OSF/1 operating system was introduced. The rest of the essential features is expected to come with Release 1.2 in mid 1994. As it is difficult to give a valid description in

such a fast changing situation, we chose to describe the Paragon system as it is specified for end 1994 and to specially mark those features which are currently still under development. All performance results refer to Paragon OSF/1 Release 1.1 as available in February/March 1994. Most measurements were carried out on KFA's 140-node Paragon XP/S 10 system.

2 System architecture

2.1 Overview

The Paragon's processing nodes are arranged in a two-dimensional rectangular grid. The memory is distributed among the nodes. The system contains nodes for three different tasks: compute nodes, service nodes, and I/O nodes. Compute nodes are used for the execution of parallel programs; service nodes offer the capabilities of a UNIX system, including compilers and program development tools, thus making a traditional front-end computer unnecessary; and I/O nodes are interfaces to mass storage or external networks. All nodes are uniformly integrated in the interconnection network. The network provides fast routing of messages using wormhole routing with a deterministic routing algorithm. The measured bandwidth between two nodes is 35 MB/s in each direction, virtually independent of the distance between the nodes. The start-up latency for a message issued by a program written in a high-level language is currently 90 μs. These values will improve to 75 MB/s and 50 μs with Release 1.2 in mid 1994 and to 175 MB/s and 30 μs by the end of 1994.

Mass storage devices and external networks are attached to special I/O nodes. Disk arrays (RAIDs) having a capacity of 4.8 GB each provide internal disk space. They are built into the Paragon cabinets; each one is connected to a single I/O node with 5 MB/s bandwidth. For external disks, tapes, and networks SCSI-1, SCSI-2, HiPPI, and Ethernet interfaces are available. A built-in Diagnostic Workstation is used for diagnostics and for booting the system. It is connected to the nodes by a separate network.

Each Paragon cabinet has a footprint of 56×107 cm, and can contain 64 nodes, each on a separate board, and up to 8 RAIDs or 6 RAIDs plus the Diagnostic Workstation. A full cabinet has a power consumption of about 5 kW; the system is air-cooled. Intel SSD has delivered Paragon systems with up to 1984 nodes.

2.2 Node architecture

Compute nodes, service nodes, and I/O nodes are all realized by the same General Purpose (GP) node hardware (cf. Fig. 1). All components of the GP node's compute and network interface parts are connected by a 32-bit wide address bus and a 400 MB/s 64-bit wide data bus.

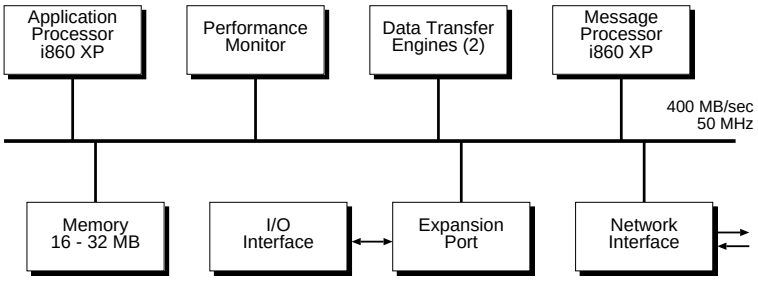


Fig. 1. Paragon node components

Compute part. The compute part includes an Intel i860 XP microprocessor, whose clock speed is 50 MHz (20 ns p/usr/local/doc/dokumente/ib/i9409er cycle), and 16 or 32 MB of memory. The memory is constructed from 4 Mbit, 60 ns DRAM chips. It is organized in two banks and has single-bit error correction and double-bit error detection. The peak speed of data transfer between the memory and the processor caches is 64 bits per cycle, i.e. 400 MB/s. The node memory can be expanded by a daughter card that can carry additional 32, 64, or 128 MB.

The i860 XP is a fast compute-oriented RISC processor in 0.8 micron CMOS technology and contains more than 2.5 million transistors (cf. Fig. 2 and [?]). Address paths are 32 bits wide, and data paths are 32, 64, or 128 bits wide. Besides the RISC core, it has two 16 KB caches, one for data and one for instructions, and two integrated vector pipelines for 32-bit and 64-bit IEEE floating-point add and multiply. Furthermore, it has an on-chip memory management unit supporting the caches and virtual memory. Page sizes are 4 KB and 4 MB; the Paragon operating system uses only the small pages.

The data cache and the floating-point registers are connected by a 128-bit wide data path, enabling a data rate of 128 bits per cycle or 800 MB/s. As the pipelines can work simultaneously and the adder can deliver one result every clock cycle and the multiplier one result every two clock cycles the theoretical peak performance of the i860 XP is 75 MFLOPS (64-bit arithmetic). For 32-bit arithmetic, also the multiplier delivers one result per cycle.

Network interface part. The interface to the interconnection network consists of a message processor, a network interface controller, and two Direct Memory Access (DMA) controllers. The message processor is a second i860 XP processor operating in parallel and sharing the memory with the application processor. Its task is to perform the details of inter-node communication and to provide global communication functions. Thus the application processor is not interrupted by message-passing operations; context switching and code turbulence and draining of the floating-point pipelines are avoided.

For outgoing messages, the message processor splits longer messages into packages, adds protocol information, and initiates the transfer. Incoming messages are autonomously received and the application processor is informed when

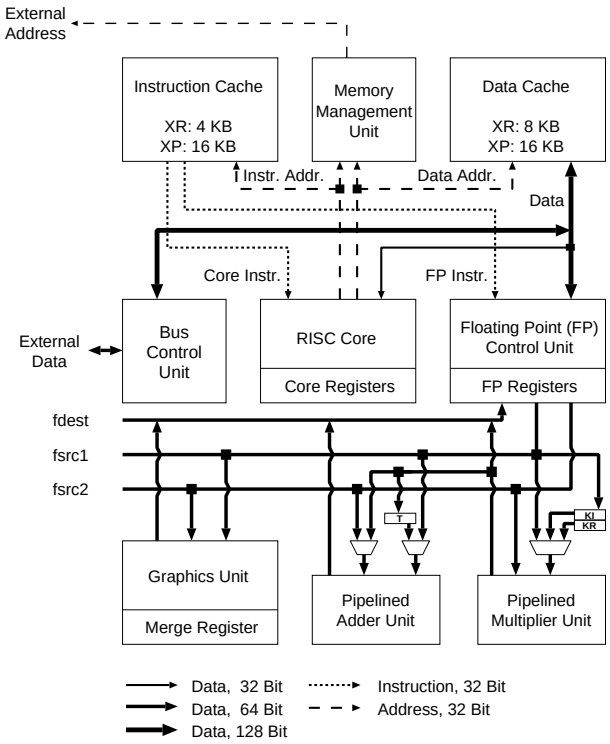


Fig. 2. Components of the Intel i860 processor (from [12])

a message has completely arrived in memory. The message processor also handles global operations independently including broadcast to and synchronization of nodes as well as reduction operations on integer, floating-point or logical operands (e.g. global sum, global minimum, global and). The message processor will be utilized as from Release 1.2 of the operating system.

The actual transmission of data between the memory and the network is effectuated by a special Network Interface Controller (NIC). It is assisted by two DMA controllers, one for inbound and one for outbound messages, which can operate in parallel.

Additional hardware. The Paragon GP node contains a data capture chip (RPM) that non-intrusively collects node performance data by monitoring bus activities. The data are transmitted to a service node through the interconnection network and are made available to the user through the System Performance Visualization Tool (SPV).

Furthermore, every GP node is equipped with an expansion port. On I/O nodes, this is used to attach a special I/O interface card.

Hardware plans. A multiprocessor node board is being designed that carries three i860 XP processors: two application processors and one message processor all sharing the node memory. It will be available and supported by the operating system by the end of 1994.

2.3 Interconnection network

The Paragon has two communication networks: the high-speed data network and the diagnostic network. The diagnostic network is used for booting and diagnostics. It conforms to the IEEE 1149.1 JTAG specification [?]. This implements serial scan strings which provide access to the various Paragon hardware components.

The data network is the main communication vehicle between all nodes. For its topology, Intel SSD has chosen a 2-dimensional mesh (cf. Fig. 3). It is constructed from Paragon Mesh Routing Chips (iMRCs) which are connected by 16-bit wide high-speed channels. To each iMRC, one node may be attached. The iMRCs can, however, route messages autonomously and are independent of the attached node, and in most Paragon systems there is a number of iMRCs which have no node attached to them. Routers and channels are combined into backplanes carrying 16 routers (4 rows and 4 columns). Four of these active backplanes accomodate the nodes of one cabinet.

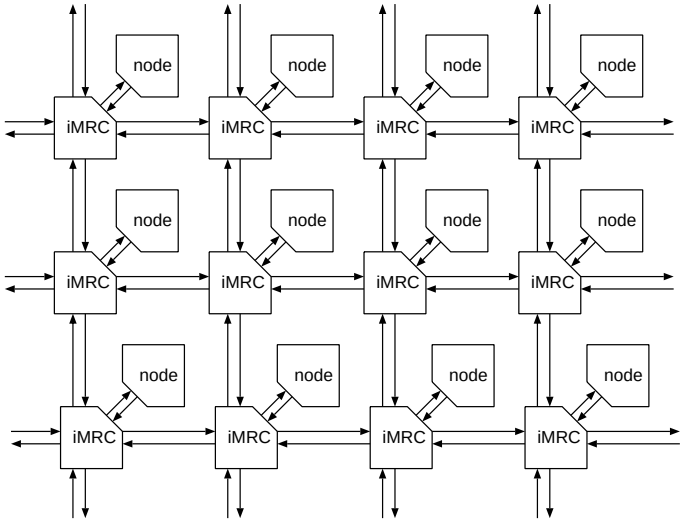


Fig. 3. Paragon interconnection network

The Paragon Message Routing Chip (iMRC). The iMRC is a message routing chip with 5 input and 5 output channels. One input and one output

channel are connected to each of the 4 neighbouring nodes (east, west, north, south). The last pair of channels is connected to the Network Interface Controller (NIC) on the attached node. A 16-bit buffer on the iMRC is assigned to every input channel. The iMRC can route a 16-bit quantity of data from an input buffer to any output channel. If the direction does not change, it takes 40 ns to make an individual routing decision and to close the resulting switches. The iMRC hardware supports broadcasting by automatically routing a message to all nodes in the rectangle between the sending and the receiving nodes.

Wormhole routing. The unit to be transmitted between nodes is one *packet*. The message processor on the node partitions the messages into packets of 8 to 1984 bytes (user controlled, default 1024 bytes); it also adds the necessary routing information to the packets and controls the transmission of packets between the network and the node.

Wormhole routing has been introduced as a fast switching technique for direct networks by Seitz and Dally [?, ?] to combine a pipelined transmission of packages with low storage requirements on the routers. In wormhole routing a packet is divided into a sequence of *flits* (flow control digits). The flit size in the Paragon system is 16 bits. A flit can be transmitted between adjacent routers in parallel in a single step.

The header flits of a package determine its path in the network. As the header advances along the specified route, the remaining flits follow in a pipelined fashion. When the header is blocked, because a channel that it wants to use is already in use by another packet, the whole pipeline stops. The sequence of flits remains in place, i.e. in the input buffers of the router chips until the requested channel is free. A packet's header thus reserves a path for its packet, and the individual channels are owned by the packet until the last flit has been transmitted.

The pipelined nature of wormhole routing makes network latency nearly independent of the distance between the sending and the receiving node, at least for longer messages. Furthermore, it requires only very small buffers on the routers. As it is difficult to synchronize a large network, synchronization is replaced by a handshake protocol between neighbouring routers. To implement such a *self-timed network* the Paragon uses a request line and an acknowledge line in addition to the data lines of each channel.

A serious problem with wormhole routing is deadlock. It can easily occur because packets travelling through the network constantly request new resources (channels) while occupying others. Deadlock is avoided by selecting adequate routing algorithms. The Paragon uses a simple but effective approach: messages are first sent in the horizontal direction and then in the vertical direction. A change of direction is allowed only once on the path. This algorithm is *minimal* (it uses a path of minimal length) and *deterministic* (as opposed to adaptive algorithms which can take collisions into account and make detours).

In detail, routing on the Paragon works as follows. A packet has two header flits containing the orientation of the path and the number of intermediate hops

to be passed. The first flit contains the orientation and the number of hops in the horizontal direction. After deciding at the sending node whether to go left or right, the flit proceeds from router to router being decremented by one at every router it passes. When its value is zero, the first header flit is stripped off, the direction is changed to vertical, and the second flit is used to determine the orientation and the number of hops to the destination.

2.4 I/O and mass storage

Specialized I/O nodes located at any free position in the network act as interfaces between the processing nodes on one side, and mass storage and external networks on the other side. Their number is in principle arbitrary and does not depend on the number of compute and service nodes. An I/O node is built from a regular GP node by plugging a specialized adaptor card into the node's expansion port.

For different I/O requirements, several types of I/O nodes are available including the MIO node, the SCSI-2 node, and the HiPPI node. The MIO node provides a SCSI-1 interface (5 MB/s, e.g. for internal disk arrays), an Ethernet interface (10 Mbit/s), and a V24 interface. The SCSI-2 node currently exists in an 8-bit, 10 MB/s version; a 16-bit, 20 MB/s version is planned for 1994. The HiPPI node (100 MB/s) consists of two physical nodes; it serves to attach external disk arrays or frame buffers as well as FDDI networks (100 Mbit/s).

Every I/O request issued by a node is serviced by an I/O node. When, for example, a program requires data from a disk file, the data are read and buffered by the I/O node to which the physical device is attached, and are transferred to the requesting node via the interconnection network. All this happens transparently to the user program.

The primary mass storage systems of the Paragon are disk arrays (RAIDs) which are integrated in the Paragon cabinets. Each disk array has its dedicated MIO node to which it is connected via the SCSI interface. The RAIDs consist of five 3.5-inch commodity disk drives with a capacity of 1.56 GB each, which hold the data and parity information. The RAID controller uses RAID level 5 functionality. In total one disk array can accomodate 4.8 GB of data. In the event of a drive failure, the Paragon operating system, together with the controller, provides routines to rebuild the data. The file system remains intact and can be used while the reconstruction is performed. Several file systems usually resident on different RAIDs can be combined to a single Parallel File System (PFS).

In addition to the internal disk systems, connections and services to support external disk storage and backup systems are also available for the Paragon, including HiPPI, FDDI, and UniTree. An FDDI network is attached to the Paragon via an NSC router connected to the HiPPI node. Furthermore, there is a QIC-150 streamer on the Diagnostic Workstation, and another internal 4 mm streamer tape can be attached to an MIO node.

2.5 Fault tolerance

The strategy on the Paragon for achieving a certain level of fault tolerance is twofold: there are several means for online diagnostics on one hand, and there are ways of concurrent repair and operation on the other hand.

A separate diagnostic network controlled by the Diagnostic Workstation monitors important system components including CPUs and memories on the nodes, iMRCs, I/O interfaces, and power supplies. The Diagnostic Workstation is also used for booting the system. Online diagnostics can test groups of nodes without disturbing the computational activities of other nodes. The interconnection network provides error detection capabilities in both hardware and the message-passing protocols.

When a fault has been detected in a node, a disk array, or a communication component, the system administrator can reconfigure the system so that part of the machine can still be used. A faulty node is marked so that it is no longer allocated to users. This does not affect the communication network, since the mesh routing chip to which the node is attached can continue operation. The system must, however, be powered down, when the node is replaced. In case of a faulty disk in an internal RAID system the disk can be replaced during operation of the rest of the machine and the contents of the disk can be rebuilt online while normal disk I/O goes on.

3 Operating system

The Paragon operating system was designed to provide an application interface compatible with the OSF/1 operating system developed by the Open Software Foundation [?], the NX message-passing interface compatible with the iPSC/860, and a parallel file system extending the Concurrent File System of the iPSC/860.

3.1 Paragon OSF/1

To realize the application interface an "Advanced Development" multiprocessor version of OSF/1, called OSF/1AD, was developed. OSF/1AD is based on the Mach 3 *microkernel* developed at Carnegie Mellon University [?]. However, to support an environment without shared memory, the NORMA (NO Remote Memory Access) version is used. The Paragon operating system architecture is shown in Fig. ??.

The Mach kernel provides threads, tasks, and ports as key mechanisms for building higher-level system services. These mechanisms are usually not directly exposed to the user. A *thread* constitutes a basic unit of execution. A *task* is a virtual address space in which a set of related threads execute with protected access to system resources. Threads can be executed concurrently with other threads, even within the same task. The conventional notion of a *process* is, in Mach, represented by a task with a single thread of control. Communication between tasks is based on a client/server system structure in which tasks (clients)

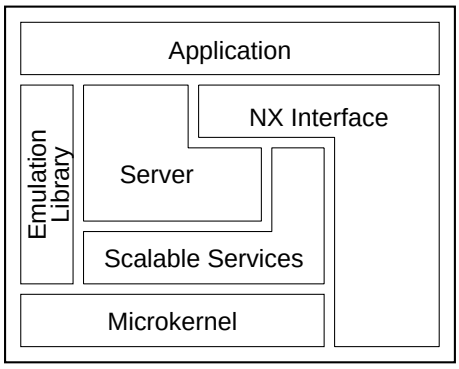


Fig. 4. Paragon OSF/1 operating system architecture

access services by making requests of other tasks (servers) via messages sent over a communication channel called *port* (Inter Process Communication, IPC). A port is a unidirectional channel consisting of a queue holding messages managed and protected by the kernel. A task holds rights to these ports that specify its ability to send or receive messages. Only one task can hold the receive right for a port.

Each Paragon node runs the microkernel that supports basic system services. An OSF/1 *server*, implemented outside the kernel, runs on every node and provides access to all OSF/1 services including process management, file system, and network access.

In addition, an *emulation library* is linked to an application, implementing those parts of the operating system that can be executed in the same task as the UNIX process. There are three kinds of operations:

- Those which can be performed completely locally to the task, such as returning the process identifier.
- Those which use Mach services directly, such as creating a new thread.
- Those which call the OSF/1 server.

These OSF/1 servers and libraries cooperate to offer the view of a single UNIX-like system to the user.

The *NX interface* provides a superset of the NX/2 message passing-interface. Applications use the fast kernel-level NX interface rather than the microkernel IPC for message-passing.

The Paragon OSF/1 supports virtual memory on all nodes.

File systems. The Paragon OSF/1 file system is based on the Berkeley 4.3 Virtual File System (VFS). VFS provides an abstract layer interface to different UNIX file system types, especially to the UNIX File System (UFS) and to the Network File System (NFS). UFS is compatible with the Berkeley 4.3 Tahoe release. On top of the VFS, which is a single-processor file system, the Distributed

File System (DFS) has been built. This provides a common logical view of the file system structure for every process on every node. There is only one global directory tree which transparently combines local file systems of RAID disks and NFS-mounted file systems.

In addition to its support for UNIX-type file systems, the Paragon operating system offers the Parallel File System (PFS), which provides file services at high data transfer rates by striping files across multiple I/O nodes and their attached RAID systems. The amount of data from a PFS file that is stored in each RAID system is determined by the stripe unit (e.g. 8 KB or 64 KB), which is set by the system administrator. There are three forms of parallelism for I/O to a PFS file. If a single node reads or writes a block of bytes that is larger than one stripe unit, the different stripe units can be read or written in parallel. If two of more nodes access different RAID systems at the same time, the disk operations can proceed in parallel. Additionally, the single disks of each RAID system operate in parallel. PFS can be used with standard OSF/1 system calls and commands. For parallel applications there are also parallel I/O system calls including those of the Concurrent File System (CFS) that exists on the iPSC/860.

The UniTree client interface is available for access to servers for file exchange as well as backup and restore.

Single system image. A single system image across the multicomputer system for management of processes, files, user authorization, accounting, etc. is realized by *scalable services* which integrate the individual services of all nodes. For example, users are able to obtain status information regarding all processes in the system using the *ps* utility, killing a process is possible wherever the process may be. The DFS allows all I/O devices to be equally accessed from any node. The system is managed as a conventional one-processor system.

3.2 Partitioning and scheduling

The Paragon operating system allows the processor mesh to be divided into sets of nodes, called partitions. Partitions provide a means of restricting access to portions of the mesh for particular users or types of jobs and a way to specify different scheduling characteristics on different portions of the machine. In principle, a partition may comprise any arbitrary set of nodes. At least the following partitions which are established by the system administrator must always be present (cf. Fig. ??):

- The root partition; this consists of all the nodes in the machine.
- The service partition; this supports general user services, such as editors, compilers and UNIX shells. Operating system services, such as the file server, are physically located on service nodes. Load levelling is supported within the service partition by automatically moving new processes to less loaded nodes.

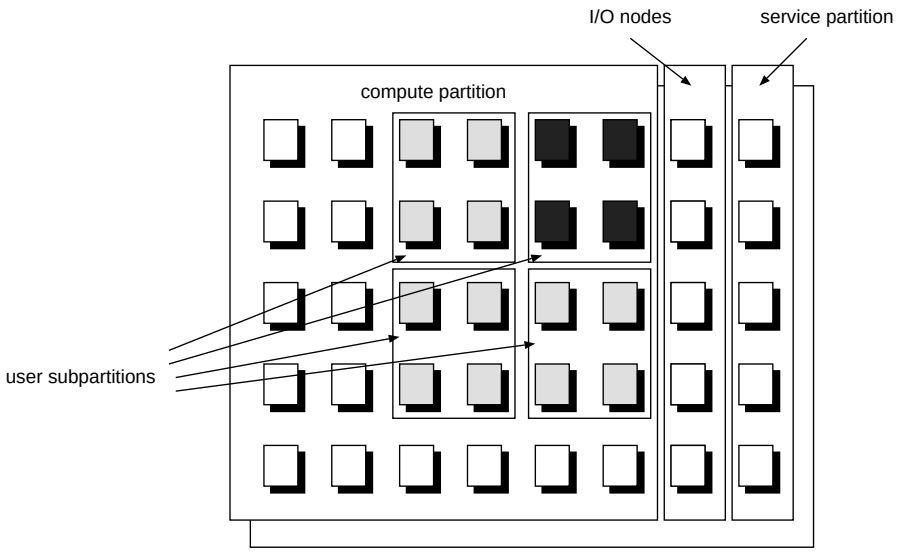


Fig. 5. Paragon partitions

- The compute partition; this consists of the majority of the nodes in the system. Here the users' parallel applications are executed. The compute partition is hierarchical; i.e. it may be divided into subpartitions, which themselves may have subpartitions and so on. Subpartitions may also overlap. Subpartitioning of the compute partition can be done by the system administrator as well as by the user.

The I/O nodes can be grouped into an I/O partition. Normally, however, these nodes are added to the service partition.

Subpartitions are defined by specifying the parent partition from which to allocate the nodes, the specific nodes to allocate, access permissions, and the scheduling characteristics of the partition. Attributes similar to the UNIX file system modes control the access to partitions (for user, group, all):

- r allows the subpartitions of a partition to be displayed,
- w allows the attributes of the partition to be changed and to create and remove subpartitions, and
- x allows applications to run in the partition.

If access permissions or scheduling characteristics are not explicitly assigned they are inherited from the parent partition.

Depending on the type of the partition, different scheduling mechanisms are available. In the service partition, processes are scheduled in the usual UNIX-style timesharing mode, in which they run for a short period of time (typically 100 ms) or until they issue a blocking system call.

Parallel applications will usually run in the compute partition and will there be scheduled according to the *gang scheduling* mechanism. In this model all the processes which make up an application are scheduled at once on all the nodes on which the application has been loaded. The application will run until the end of its roll-in quantum, at which time the system will determine whether there is another application with equal or higher priority ready to run. As the paging facilities of the operating system are used to page an application in and out, small applications need not be moved between disk and memory on a context switch. The roll-in quantum is an amount of time specific of the partition and will be in the range of several minutes. Applications in overlapping partitions are always scheduled in a way such that only one application runs on any single node at one time. Gang scheduling will become available in mid 1994.

When on a single node more than one process is associated with an application, this set of processes is scheduled by the normal UNIX scheduling, while the application is active. Applications are not rolled out, when they issue system calls. Therefore asynchronous, i.e. non-blocking, I/O is provided.

3.3 System access and accounting

The user can develop and run parallel applications on the Paragon system either interactively with remote login facilities or by submitting batch jobs. The Paragon appears as a stand-alone UNIX system on the network connected to his workstation. After login, the user is running his shell somewhere in the service partition. Although the Paragon requires no front-end, neither for program development nor for system administration, most program development tools including compiler and performance analysis tools are available as cross development versions running on SUN and SGI workstations.

The Paragon utilizes the Multi-User Accounting and Control System (MACS) developed by the San Diego Supercomputer Center to manage the system resources and the Network Queueing System (NQS) developed at NASA Ames to manage batch jobs. It provides flexible, automated job scheduling schemes for assigning system resources. MACS allows simultaneous batch and interactive scheduling and control, using separate partitions for each. The scheduler allows jobs to be executed from the NQS queues. The system administrator specifies for each queue the number of nodes, the priority, and the number of jobs permitted to wait for execution. MACS can monitor and account for system resource usage, e.g. number of nodes and execution time of the job, producing data for analysis and reporting purposes. MACS includes facilities for automatic or selective preemption of jobs that have exceeded their resource allocations.

3.4 Message-passing

Processes in parallel applications use several facilities both in hardware and in software to exchange information. As mentioned earlier, these include the message processor, a second i860 processor on every node. The message processor

shares memory with the application processor, and the two processors communicate with each other via shared variables. When the application processor wants to make a message-passing call, it places the parameters into these variables. The message processor regularly polls them and executes the call when it finds the information. The basic message-passing software, which, e.g. packetizes the messages and controls the transfer, runs on the message processor at the kernel level and can directly address the hardware. Thus, the involvement of the OSF/1 operating system in message-passing is minimal keeping the startup latency low.

Outgoing messages are sent directly out of the process memory. For incoming messages, a system buffer is provided on every node. It contains a distinct buffer for each node in the Paragon system. Every sending node knows the status of its associated buffers on all receiving nodes. A process sends the first packages of a message into this buffer. As soon as a receive is posted, the message processor will transfer the data into the process memory, thus freeing space for subsequent packages. It will also inform the application processor when the message has completely arrived by setting a shared variable. If a receive has been posted before the message arrives, the packets go directly to the process memory. The various buffer sizes can be selected by the application. The message processor will be used in this way as from mid 1994.

4 Program development environment

Diverse programming models are supported by the Paragon system's development environment. A broad range of programming languages, optimized mathematical libraries, and a set of tools assisting the user to create new applications or to port existing codes are available from Intel SSD or as third party products. Most tools can be used on the Paragon service nodes or as cross-development tools on Sun and SGI workstations. Several components of the Paragon system's development environment have been ported from the iPSC/860.

4.1 Programming models

The Paragon system supports several programming models. Besides the message-passing model, which is basic for most distributed-memory multicomputers and is also the Paragon's primary programming model, the Paragon supports the data parallel model through High Performance Fortran (HPF) and the shared-memory model through Shared Virtual Memory (SVM).

Message-passing. In this programming model independent processes are running asynchronously. They communicate by explicit message-passing. Messages are also used to synchronize processes. On the Paragon, several processes belonging to one application can run concurrently on one compute node.

In principle, the processes can be totally different programs. The most common programming style, however, is the *Single Program Multiple Data (SPMD)* style where the same program runs on each node allocated to the application,

but each node works only on its part of the data. For *perfectly parallel problems*, each process can do its work without access to data held by other processes. For other types of problems, the processes must exchange data with each other to do their work. Another popular programming style is the *manager/worker* concept. One manager process starts several worker processes and assigns them their tasks. As soon as a worker process has finished its task it reports to the manager process which accepts and interprets the results and assigns it a new task.

Data parallel. For the data parallel programming model, Intel SSD will provide High Performance Fortran (HPF) [?], an extension to Fortran 90. Parallelism is expressed in the program by array operations. Data distribution directives describe how arrays are to be distributed to the parallel processes. The programmer also specifies a mesh of processes which is then mapped to the real hardware. The compiler takes responsibility for inserting the explicit communication instructions required for running the code on a distributed-memory system. The amount of communication and load balancing is determined by the mapping of data to processes. Each process is responsible to perform the computation for its assigned data. Therefore, the programmer can control the communication costs and load balancing with the help of the distribution directives. HPF provides two important benefits for the parallel programmer: a familiar programming model and portability by machine-independent specification of the data distribution.

Shared memory. To a limited extent, the Paragon system also incorporates Shared Virtual Memory (SVM) [?] that allows building parallel applications in which data are logically shared between processes on one or more nodes. The distributed physical memory forms a uniform global address space accessible from every node. There is no need for explicit message-passing. On the Paragon, SVM can be used via the standard UNIX System V shared segment interface. Shared segments can be mapped into the virtual address space of different processes.

4.2 Message-passing libraries

The message-passing programming model is widespread, and many message-passing libraries have been developed. Some of these are related to a special distributed memory machine, e.g. Intel SSD's NX message-passing library. Others are portable in a sense that they allow the user to specify processes and inter-process communication in a machine-independent way.

NX message-passing library. The Paragon OSF/1 operating system includes the NX message-passing library which is known from the iPSC series. Some extensions are supported on the Paragon including the possibility of allocating more than one process of a parallel application to a node. Synchronous (csend,

crecv) and asynchronous (isend, irecv) messages, as well as messages producing interrupts (hsend, hrecv) are supported. Additionally, global operations for performing operations that use data from every node, e.g. for a global sum, are available.

Other message-passing libraries. Portable message-passing libraries on the Paragon include PVM of Oak Ridge National Laboratory, EXPRESS of Parasoft, PARMACS of the Gesellschaft für Mathematik und Datenverarbeitung (GMD) and Pallas, P4 and TCGMSG of Argonne National Laboratory, and MPI based on P4.

4.3 Languages and compilers

The Paragon system offers a set of programming languages which are interoperable, allowing compiler output to be linked irrespective of the source language. Compiler switches allow different code generation strategies (e.g. scalar code, software pipelined loops, and vector code) to be selected, as well as different optimizations (e.g. scalar optimization, loop transformation, and cache management). The compilers exploit the advanced hardware features of the i860, such as dual instruction mode, dual operation instructions, and the arithmetic and load pipelines. The compilers were originally developed for the iPSC/860 and have been adapted to the Paragon.

Fortran 77 and High Performance Fortran. Version 4.5 of the Paragon Fortran 77 compiler for Paragon OSF/1 is available. Compilation can be performed on the service nodes and on workstations using the cross compiler.

The FORGE HPF batch pre-compiler, called xhpf, is available from Applied Parallel Research. It includes MAGIC, an optional automatic parallelization mode. xhpf with MAGIC takes as input a serial Fortran 77 program and automatically generates a parallelized code with Fortran 90 array syntax and HPF directives. From this, xhpf produces a Fortran 77 program with embedded calls to a communication library.

C, C++, and Ada. Version 4.5 of the Paragon optimizing C compiler for Paragon OSF/1 is available on the service nodes and as a cross compiler. For object oriented programming Paragon C++ is available which is based on the AT&T cfront preprocessor which translates C++ into C. For the installation of this compiler a license of AT&T is necessary. An Ada compiler for Paragon OSF/1 is available, too.

4.4 Program development tools

The tools available on Paragon are based on the Tools Application Monitor (TAM). This is a per-node, per-application server for tools that perform application process monitoring on the Paragon system. Figure 6 shows the co-operation of the program development tools.

All graphical tools described below can be started by the graphical user interface ParAide which also assists the user in loading parallel applications on the compute nodes and provides a means to open text files into an editor.

Debugging. The Interactive Parallel Debugger (IPD) is a source-level debugger for large parallel application programs written in Fortran, C, or Assembler. The IPD allows context debugging to access and control selected groups of processes spread across multiple nodes. The IPD has a data reduction mechanism and facilities to examine the message-passing events and structures. Breakpoints can be set in some or all of the application's processes. An enhanced version, the XIPD supports a graphical user interface based on Motif. XIPD provides continuous update of node status, indicates which routines are currently executing, and notes where the execution point is in the code.

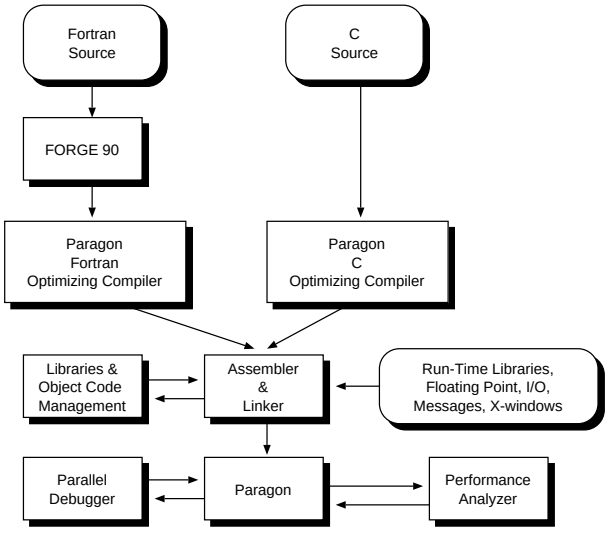


Fig. 6. Paragon program development

Performance analysis. The runtime profiling tools prof and gprof, which are special versions of the UNIX profiler prof and gprof for the Intel i860 processor, can be used to analyze an application program. During a program run, the IPD can collect profiling data on every node. These are presented in tables and provide the user with information about the number of subroutine calls and the execution time of these routines. prof produces a simple execution profile, whereas gprof additionally produces a call-graph and a cycle listing. XProf and XGprof provide graphical front ends to prof and gprof.

To monitor the performance of the Paragon system Intel SSD has developed the System Performance Visualization Tool (SPV). It allows the user to display the Paragon front panel lights on his workstation and get information on CPU, mesh, and memory bus utilization. The hardware performance monitor RPM on each compute node collects mesh and memory bus information while the Mach Kernel collects CPU idle-time information, page fault rate, etc. Each second these performance data are sent to the spv daemon running on the service partition. Data transfers are optimized by using a logical tree structure. The spv daemon sends this data to every spv client running on either the native Paragon or on a remote workstation.

For software-based performance monitoring of applications, the Paragon offers a tool built on the ParaGraph display system developed at the Oak Ridge National Laboratory. This system presents an animation of the execution of parallel applications as derived from trace information gathered during program execution. In addition, the user can request graphical summaries and statistical analyses of overall program behaviour in a variety of display formats. ParaGraph which uses Motif replaces the Performance Analysis Tools (PAT) which are provided on the iPSC/860 for identification of time intensive parts in an application program.

Utilities. A parallel make utility, pmake, that maintains up-to-date versions of target files and performs shell programs in parallel, is available on the Paragon. It is an extension of GNU make. The pmake command updates multiple target files in parallel. Parallel execution may occur either in the service partition or the compute partition. In the service partition, pmake relies on process migration and load balancing to ensure efficient parallel execution. In the compute partition, pmake places commands on the available nodes within a partition and executes as a parallel application.

4.5 Optimized mathematical libraries

To ease the process of porting software to the Paragon and to provide means of exploiting the machine's computational power, Intel SSD offers several libraries of mathematical routines, both node libraries with sequential routines and parallel libraries.

Node libraries. As the Paragon Fortran compiler does not always produce code that makes the most of the i860 processor, it is mandatory in order to achieve high node performance to apply optimized library routines, at least for the mathematical kernels of the applications.

The main library for this purpose is the Basic Math Library, an implementation for the i860 processor of the CLASSPACK Basic Math Library from Kuck and Associates. It contains the Basic Linear Algebra Subroutines (BLAS) and also several FFT routines and solvers for tridiagonal and pentadiagonal linear systems. The Basic Math Library forms the basis for other libraries such as the

commercial NAG Library or the public domain Linear Algebra PACKage (LAPACK) [?] that contains routines for the solution of dense linear systems and eigenvalue problems.

The Signal Processing Library (SEGlib) includes a collection of signal processing routines that are compatible with the SEGlib Seismic Subroutine Standard Library.

Parallel libraries. Intel SSD has developed the ProSolver software package for the solution of large systems of linear equations. The matrices can be stored either on disk or in local memory. There are three separate products: ProSolver-DES applies a direct method to dense matrices, ProSolver-SES is a skyline direct solver for sparse matrices, and ProSolver-IES is an iterative solver for general sparse matrices.

There is a growing collection of mathematical software developed for iPSC and Paragon systems by institutions outside Intel SSD. References to this software can be found in the software catalog [?].

4.6 Visualization tools

The Paragon system provides several levels of support for network-transparent, client/server visualization. The X Window System X11 Release 5, PEX, and Motif are included in the system software. These tools enable client applications to run on the Paragon system and direct their output to a graphics workstation server. Additionally, the Distributed Graphics Library (DGL) is available for interactive graphical viewing of application results.

5 Performance

In contrast to the previous chapters that contain a description of the Paragon hardware and software and can only give upper limits for its performance, this chapter attempts to give the reader an idea of the performance that real programs can reach on the system. The first two sections report measurements of the basic message-passing and I/O performance, while the last two sections deal with computational performance and describe measurements carried out on some algorithmic kernels and on a typical application.

Most performance results were obtained on KFA's Paragon XP/S 10 system, which is equipped with 140 compute nodes (with 32 MB memory each), 4 service nodes, 1 HiPPI node, and 6 MIO nodes (16 MB) serving one RAID system each. Four RAID systems are configured to form two Parallel File Systems, one with stripe size 64 KB, and one with 8 KB.

All measurements were made under Release 1.1 of the Paragon OSF/1 operating system that does not yet make use of the message processor on the nodes. All message-handling is done by the application processor and this degrades

message-passing performance for send operations, e.g. by instruction cache loading and re-loading on the application processor, as well as for receiving operations, e.g. for asynchronous receives. This deficiency can of course considerably reduce the performance of parallel applications. It will be overcome by Release 1.2 of the operating system when the message processor will finally be dedicated to communication tasks.

5.1 Network performance

The basic parameters for the network performance are the message latency and bandwidth between the nodes. Our measurements show that the startup time for zero-length messages with synchronous message-passing is 90 μs, the maximal bandwidth between two nodes reached from an application program is approx. 35 MB/s which is 20 percent of the peak bandwidth of a communication channel; half of this achievable bandwidth is reached with a message length of 6,200 B. Fig. ?? shows the bandwidth as a function of the message length for up to 100,000 B. The hop overhead for long-distance message-passing is neglectable if the communication channels along the path used during the communication have no contention.

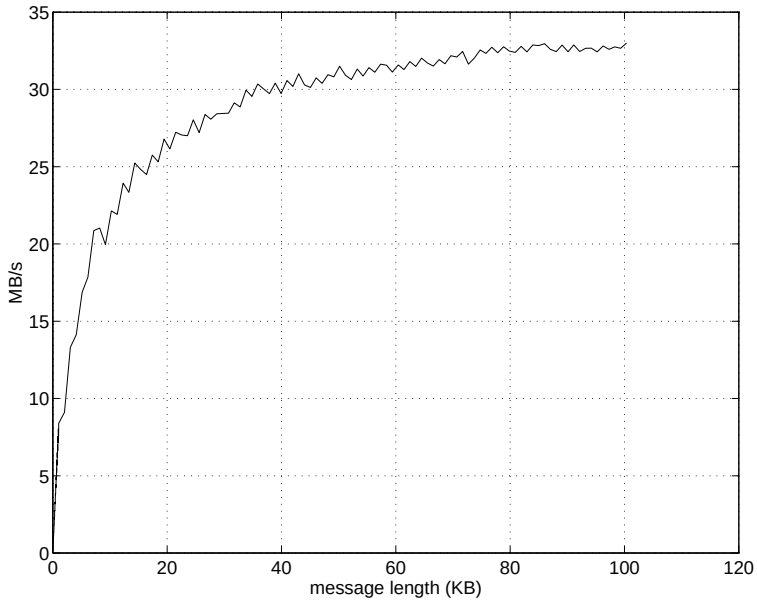


Fig. 7. Measured bandwidth between two neighbouring nodes

Synchronization between multiple nodes is not supported by the Paragon hardware; instead, it is implemented via low-level message-passing calls. The

same applies to global operations, e.g. to reductions or global sums. As the startup cost for small messages is relatively high, the performance of the global functions is dominated by this overhead; this holds especially for those communication operations which produce a large amount of small messages, e.g. global synchronization of nodes. Typically the cost of these operations grows logarithmically with the number of nodes as they are performed in a hierarchical tree-like manner. Fig. ?? shows the performance of global synchronization of n nodes while Fig. ?? shows the performance of a global summation, the latter representing a combination of communication and computation.

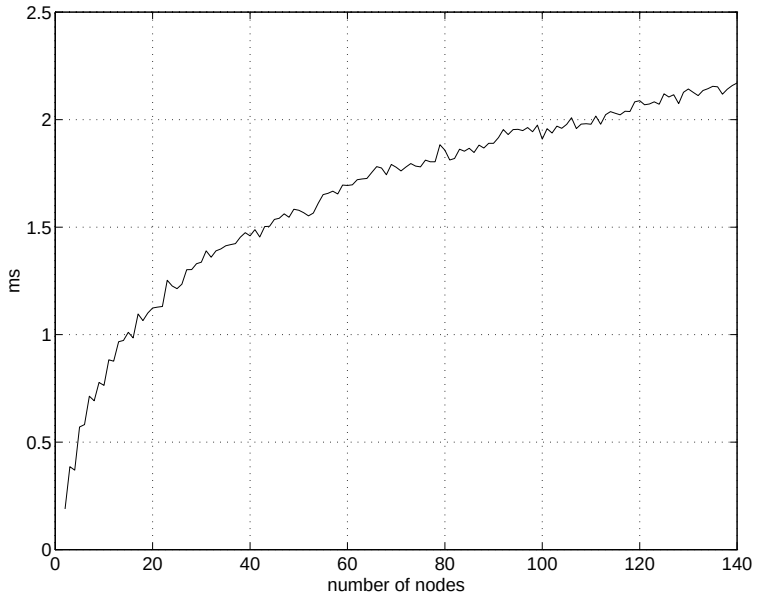


Fig. 8. Cost of global synchronization

5.2 Parallel File System I/O performance

Functionality and performance of the Parallel File System (PFS) depend on a number of parameters, partly predefined by the Paragon OSF/1 system administrator and partly selectable by the application programmer. In the following, a description of those parameters that are most important for the end-user and of their effect on performance will be given.

The current configuration of KFA's Parallel File System uses four striped RAID disk arrays (striping factor 4) with a stripe size of 64 KB. With this configuration, a user may obtain an I/O performance in the range from 0.20 MB/s to 23.08 MB/s (or virtually even from 0.20 MB/s to 116.04 MB/s, if certain redundancies in the data can be exploited) depending on the choice of I/O parameters.

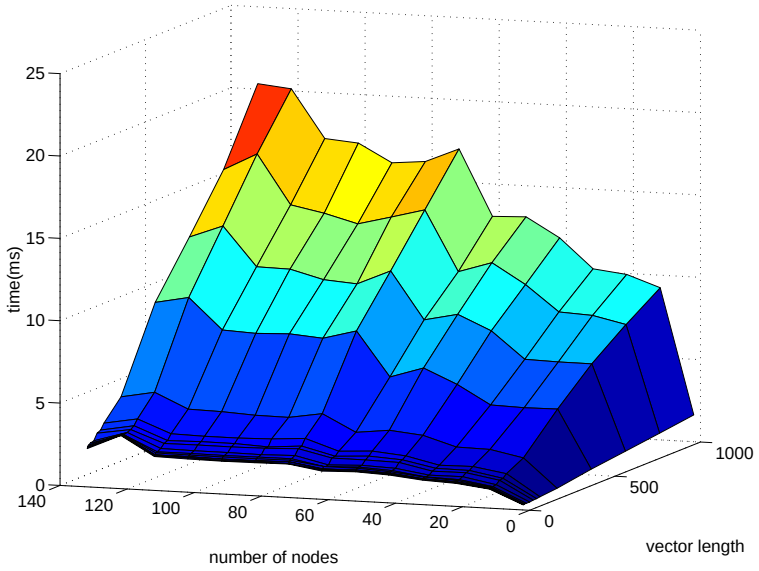


Fig. 9. Cost of global summation

The following list gives a short explanation of the considered parameters. Values actually tested in the measurements are given in brackets.

nodes [1,2,4,8,16] The number of compute nodes used for parallel I/O operations contributes to performance in that multiple communication paths to multiple I/O nodes can be used in parallel.

mode [0,1,2,3,4] There are five different I/O modes controlling the strategy of file pointer administration on multiple compute nodes (shared or private), the layout of data on disk (in node order, chronological order, or unordered), the characteristics of I/O operations (synchronized or unsynchronized), and the exploitation of data redundancies (I/O on only one or all nodes).

lrecl [8000,65000,500000,1000000,8192,65536,524288,1048576 B] The number of bytes read or written in each single I/O request ("logical record length") determines the efficiency of each request. PFS has been designed for rather large requests, multiples of the stripe size are especially honored.

size [1 MB, 20 MB per compute node] The total size of a sequence of related I/O requests is relevant, because data are buffered on the I/O nodes. Hence, data volumes that fit into the I/O buffers can be handled more efficiently, possibly even without real disk traffic.

read/write Even though not arbitrarily at the user's disposal, the distinction between read and write operations may be of importance with respect to performance.

An assessment of the importance of each parameter and the determination of optimal parameter values can be achieved by running an I/O benchmark with

all possible combinations of typical parameter values. This benchmark results in a table representing a discrete performance function over the domain given by the cross product of all parameter values. Each row of the table contains a valid parameter value combination together with the achieved I/O performance. For the evaluation of the data, the table is sorted by ascending performance values. Thus, beneficial parameter sets are found in the upper part of the sorted table, unprofitable value combinations in the lower part.

A first observation is that the performance ranges from 0.20 to 116.04 MB/s for the parameter domain given in brackets in the above list. The very high performance range from 23.65 to 116.04 MB/s is achieved only with mode 4. In this mode, only one compute node really performs I/O, all other nodes just wait for completion. On reads, the data read on one node are broadcast to the other nodes via NX message-passing, on writes, no data exchange whatsoever between the I/O-handling node and the other nodes is done. Thus, the measured I/O performance appears to be much higher than it physically really is.

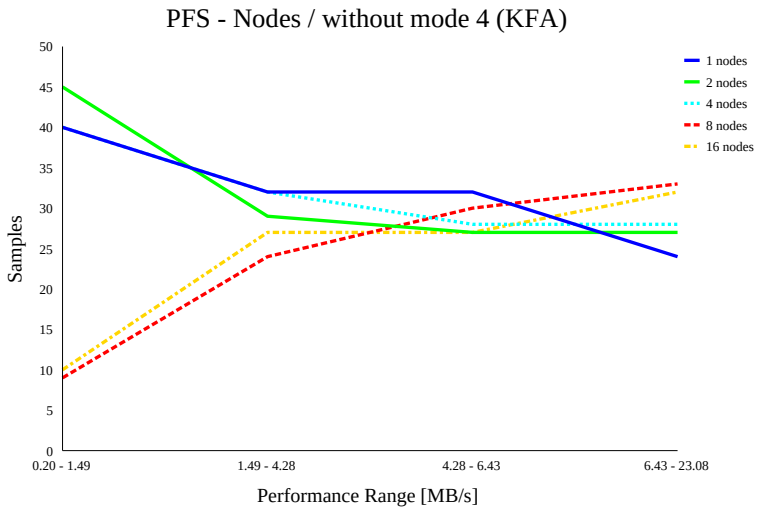
To avoid undesired interference in the evaluation, mode 4 has been excluded from all further measurements (i.e. all table entries that contain mode 4 have been deleted). With this modification, the performance ranges from 0.20 MB/s to 23.08 MB/s. This performance range is still beyond the physical bandwidth of the SCSI ports due to buffering effects for small I/O size (1 MB per node). Neglecting table entries with small I/O size reduces the bandwidth to an upper limit of 12.63 MB/s. However, only seven table entries account for high bandwidth above this limit due to buffering effects. Hence, parameter combinations with small I/O sizes have not been excluded from the further investigations.

For a more detailed assessment of the importance of certain parameter values or value combinations, the sorted performance table is now divided into four equally sized "buckets" with consecutive table entries. Bucket 1 represents parameter combinations yielding low I/O performance (0.20 to 1.49 MB/s), bucket 2 yields 1.49 to 4.28 MB/s, bucket 3, 4.28 to 6.43 MB/s, and finally, bucket 4 best performance in the range from 6.43 to 23.08 MB/s. The frequency of the occurrence of any parameter value in each bucket may now be determined. Parameter values that frequently occur in buckets 1 or 2, but rarely in buckets 3 or 4, are those that contribute much to bad performance, those found mostly in buckets 3 and 4 but rarely in 1 and 2 are, on the other hand, those contributing to high performance.

Figures ??, ??, and ?? depict the frequency distribution of the measured parameter values. In the following, some major effects recognizable in the diagrams shall be discussed.

A first observation is that there are hardly any parameter values that guarantee for good or bad performance. Most parameter values, instead, may occur in any performance range (bucket), even though with varying frequency. An exeption to this rule are record lengths 8000 and 8192 B which are exclusively found in buckets 1 and 2 and thus enforce very poor performance.

In the "IOmode" diagram (Fig. ??), only mode 3 shows a clear tendency towards good performance. It is worthwhile to notice that the default mode 0 is



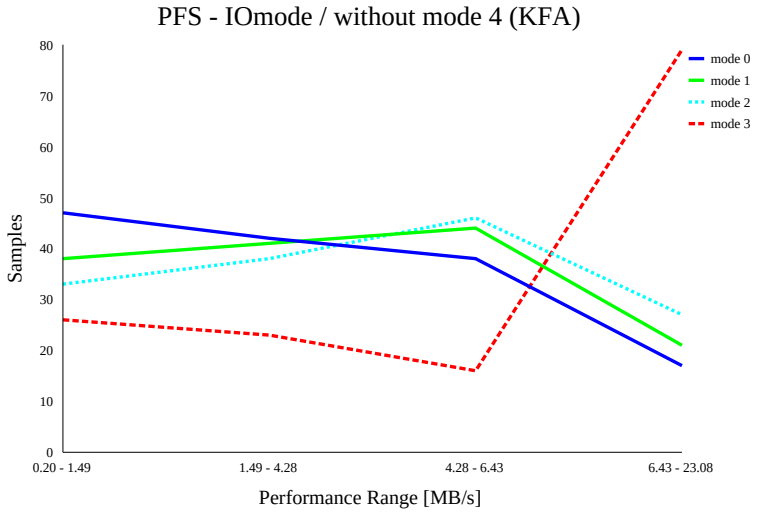
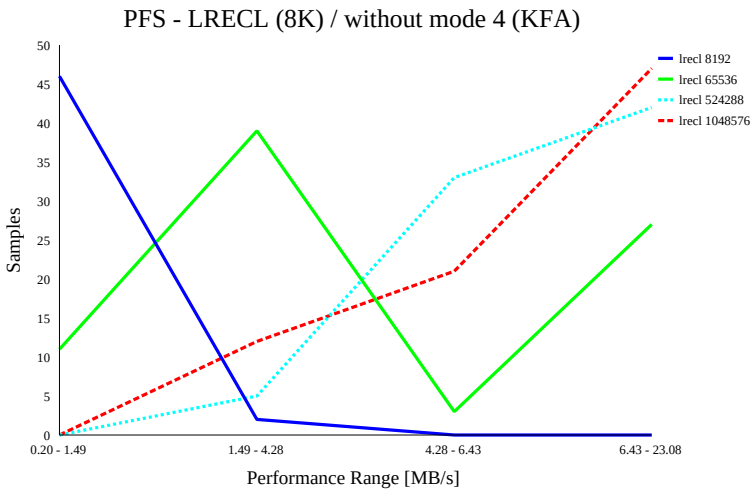


Fig. 10. Frequency distribution of PFS I/O parameters (I)

worst according to this evaluation.

Clearly, the number of compute nodes used contributes to performance. However, there is no number of nodes that ensures optimal performance. On the other hand, the "Nodes" diagram (Fig. ??) shows that it is much more likely to obtain bad performance, when using 1, 2, or 4 compute nodes than with 8 or 16 nodes.

In contrast to the already mentioned short record lengths, record lengths 524288 and 1048576 B are candidates for good performance (Fig. ??). They are never found in bucket 1 and only rarely in bucket 2. It is important to notice that



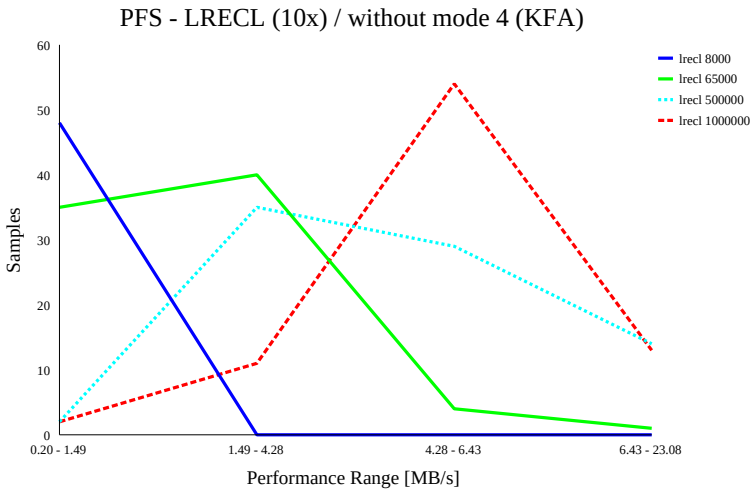


Fig. 11. Frequency distribution of PFS I/O parameters (II)

a minor change in record length towards 500000 and 1000000 B, respectively, significantly deteriorates the situation. This indicates that I/O requests with lengths that are multiples of the stripe size are handled more efficiently.

This effect can also be seen from the "I/O size" diagram (Fig. ??). In contrast to the expectations, the major criterion for good performance is not the total request size (1 MB or 20 MB). Instead, it is most important, whether or not the size is a multiple of the stripe size[1]. In fact, when summing up the frequency

[1] The total I/O size is directly related to the logical record length, in that multiples

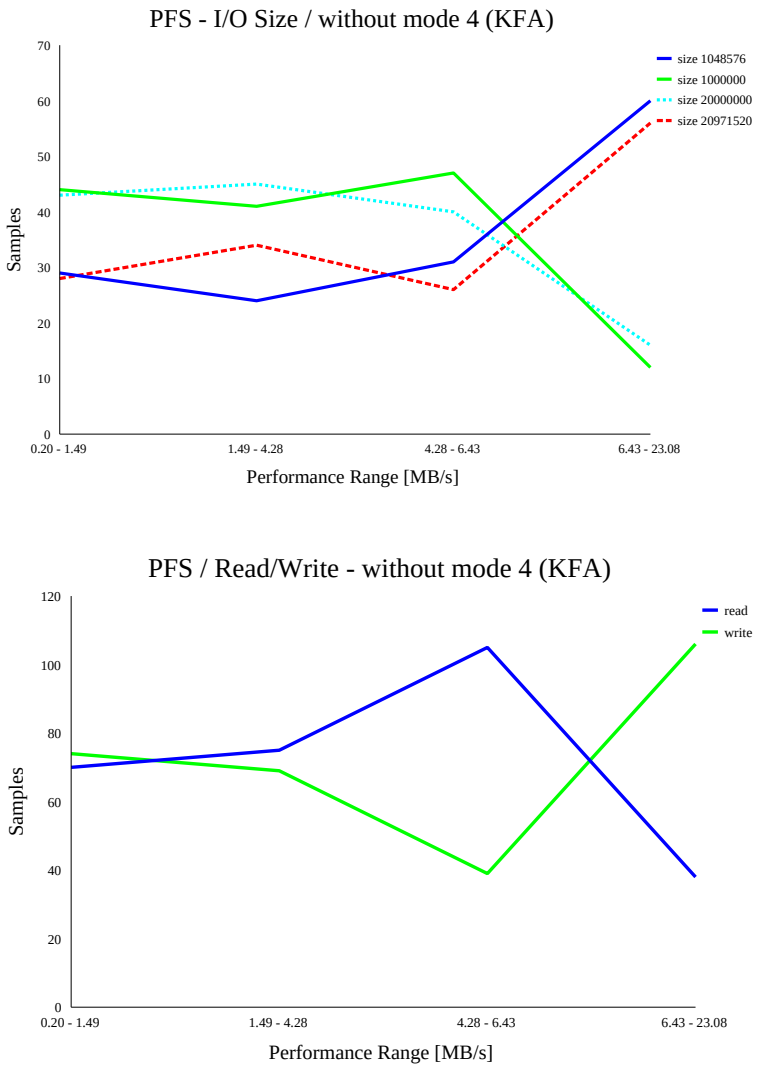


Fig. 12. Frequency distribution of PFS I/O parameters (III)

values for the two "small sized" curves or for the two "big sized" curves, no tendency towards better or worse performance is visible for one of the two cases.

The second diagram of Fig. ?? shows the performance characteristics of read and write operations. Apparently, reads and writes are found with almost equal frequency in buckets 1 and 2, i.e. if the I/O performance is poor, this is not directly caused by read or write operations, but rather by other parameters. If,

of the stripe size in record lengths always lead to multiples of the stripe size in the total I/O size and vice versa.

on the other hand, the performance is close to optimum, this is more likely to be caused by write operations (probably because write operations finish when the data arrives in the I/O node buffers, rather than on disk, whereas reads generally have to fetch the data from the disks).

Concluding, in order to verify our findings, we shall compare the performance of optimal and worst-case parameter combinations as obtained from the above investigations. According to the diagrams, an optimal parameter combination should be a write operation using 8 compute nodes, I/O mode 3, record length 1048576 B, and a total request size of 1 MB (small). The I/O performance obtained for this parameter set is 16.9 MB/s and is found at position 4 from the top of our sorted performance table. Actually, the best parameter combination yielding 23.08 MB/s differs only in that it uses record length 65536 B rather than 1048576 B, which one would not immediately expect from the diagrams, but is still reasonably possible.

For the worst-case parameter set (write, 2 nodes, mode 0, lrecl 8000, size 1000000) the situation is similar. This combination results in an I/O performance of 0.30 MB/s and is found at position 29 from the bottom of the table. Indeed, this is an even coarser approximation of the position 1 parameter set (write, 4 nodes, mode 2, lrecl 8000, size 1000000) yielding only 0.20 MB/s I/O performance. However, according to the diagrams there are many candidates for bad performance and, thus, this result is still reasonable.

5.3 Numerical performance

As long as parallel libraries are not yet available or implemented on the Paragon (cf. the paragraph concerning the ProSolver), performance results about library software can only refer to single node performance. The upper limit for the performance rates, set by the peak performance of the i860 XP node processor of the Paragon, is 75 MFLOPS for 64-bit arithmetic, corresponding to Fortran double-precision.

BLAS. The Basic Linear Algebra Routines (BLAS) are widely used in dense numerical linear algebra programs. As they provide the most important vector and matrix operations, the use of a version tuned to the specific machine provides a simple way for speeding up existing numerical programs and writing new ones. The BLAS routines also enhance portability, since programs can contain standard calls to a widely available set of library routines. The BLAS routines were developed in three groups. The BLAS 1 routines consist of vector-vector-operations, while the BLAS 2 contain matrix-vector-operations, and the BLAS 3 provide matrix-matrix-operations.

For the Paragon a specially tuned version of the BLAS is part of the CLASSPACK Basic Math Library by Kuck & Associates. The following performance results are due to the Paragon Basic Math Library Performance Report [?]. The tables show MFLOP rates versus vector length N for selected double-precision routines.

N	DAXPY	DDOT	DROT	DSCAL	DSDOT
100	19.3	28.4	33.1	14.7	31.3
200	21.4	30.4	34.7	17.0	49.7
300	22.2	32.1	35.2	17.2	57.1
400	22.0	34.7	35.4	18.2	60.9
500	22.3	35.5	35.5	18.3	62.6
1000	22.7	42.0	33.7	19.0	70.5
1500	22.8	44.6	20.2	12.0	81.1

DAXPY computes vector times scalar plus vector, DDOT computes the dot product of two vectors, DROT applies the Givens transformation matrix G to the 2 by N matrix, DSCAL multiplies a vector by a scalar, and DSDOT computes the dot product of two single-precision scalar vectors, performing the summation in double-precision.

N	DGEMV	DGBMV	DSYMV	DSBMV	DSPMV
8	3.2	2.5	1.7	2.8	1.9
16	7.4	4.3	3.2	4.6	3.5
32	14.2	5.7	6.8	5.8	7.1
64	24.0	6.7	13.2	6.7	13.3
128	32.6	9.2	21.4	9.7	22.4
256	38.9	12.0	29.1	11.6	30.7
512	41.0	13.7	35.5	12.9	36.3

DGEMV, DGBMV, DSYMV, DSBMV, and DSPMV perform different types of matrix-vector operations.

N	DGEMM	DSYMM	DTRSM	DTRMM	DSYRK
8	8.4	3.8	1.6	2.3	3.3
16	8.4	8.3	3.8	4.9	7.7
32	20.8	19.1	16.5	15.5	15.2
64	41.5	29.9	26.0	29.8	31.4
128	45.1	37.8	37.0	38.1	39.7
256	45.8	41.0	41.3	41.4	42.5
512	45.9	43.3	43.7	44.0	44.2

DGEMM, DSYMM, and DTRMM perform matrix-matrix operations, DTRSM solves a matrix equation, and DSYRK performs a matrix rank k operation.

LAPACK. The goal of the LAPACK project [?] was to design and implement a portable linear algebra library for efficient use on high performance computers. The library is based on the widely used LINPACK and EISPACK packages for solving linear equations, eigenvalue problems, and linear least squares problems, but extends their functionality in a number of ways. The main method used in LAPACK for making the algorithms run faster is to restructure them in a way that they perform block matrix operations (e.g. matrix-matrix multiplication)

in their inner loops. The block sizes can be adjusted to exploit the memory hierarchy of a specific architecture. Furthermore, LAPACK uses BLAS routines whenever possible.

The library was implemented on the Paragon by Intel SSD using LAPACK Version 1.0b (though Version 1.1 has been available since Oct. 31, 1993). It was compiled using optimization level -O3. Higher optimization levels caused too much trouble. During the tests at KFA, using the test routines from the xnetlib server, one routine could even crash the machine. For the timings, the original LAPACK timing routines from the xnetlib server were used.

N	DSYTRD	DSTEQR	M,N	DGEBRD	DBDSQR
50	8.8	7.1	50,50	1.5	6.5
100	15.0	7.7	50,100	14.0	6.6
200	20.0	8.3	100,50	21.0	7.0
300	26.0	8.5	100,100	19.0	7.2
400	29.0	8.6	100,200	19.0	6.9

DSYTRD reduces a symmetric/Hermitian matrix to real symmetric tridiagonal form by an orthogonal/unitary similarity transformation, DSTEQR computes all eigenvalues and eigenvectors of a real symmetric tridiagonal matrix, using the implicit QL or QR algorithm, DGEBRD reduces a general rectangular matrix to real bidiagonal form by an orthogonal/unitary transformation, DBDSQR computes the singular value decomposition (SVD) of a real bidiagonal matrix, using the bidiagonal QR algorithm.

NAG. The NAG Fortran library is a comprehensive collection of Fortran 77 routines for the solution of numerical and statistical problems. The version of the library implemented on Paragon is Mark 15; only double-precision mode is available. The implementation was performed by Intel SSD. Though this was done using Paragon OSF/1 Release 1.0.1 and f77/Paragon Release 4.0.6, the implementation should be appropriate also for the current OSF/1 and f77 releases (Intel communication). Most routines have been compiled with -O4, some with -O1 to avoid trouble. Some NAG programs are identical with BLAS routines; these have been removed from the NAG library in order to link the BLAS routines.

N	F03AEF	F04AFF	F02AAF	F03AAF
64	1.3	1.5	4.7	5.1
128	6.6	17.6	15.5	6.6
192	20.5	28.1	21.5	7.4
256	27.5	27.5	28.3	7.1
512	32.5	32.5	47.0	6.6

F03AEF does a Cholesky factorization, F04AFF an LU factorization; F02AAF computes eigenvalues of symmetric positive definite systems, F03AAF of non-symmetric ones.

ProSolver. This package of distributed-memory Fortran routines, developed by Intel SSD, consists of four parts. The ProSolver-SES (Skyline Equation Solver) library contains high-level parallel routines for defining or assembling, factoring, and solving large double-precision real and double-precision complex sparse systems of equations, the ProSolver-DES is the analogue for large dense systems. The ProSolver-IES part is a library of parallel routines that use conjugate gradient methods to solve large sparse systems of linear equations. For this iterative solver, which has been newly developed for the Paragon, a special distributed matrix interface was designed including routines for basic manipulations of distributed matrices. The fourth part at last, contains routines for the computation of two- and three-dimensional Fast Fourier Transforms.

Referring to Intel announcements, the dense solver is expected to achieve 35 MFLOPS, the skyline solver 20 MFLOPS per node.

5.4 Application performance: crystal growth simulation

This section outlines the performance of a crystal growth simulation program. This application was developed at KFA [?] for the optimization of the silicon production process. For the quality of the silicon crystal a constant convection in the melt is very important. The convection results from the heating, the rotation of the crucible, and the rotation of the crystal. The convection is modeled by a set of partial differential equations and determined by an explicit finite difference scheme.

The simulated crucible has a radius of 3 cm and a height of 4 cm and is discretized into 30 x 90 x 40 elements. This determines the shape of the main arrays to be 32 x 92 x 42 including some additional boundary cells. The boundary cells determine the boundary conditions at the surface of the crucible, e.g. the temperature of the heating, and the values at the inner surface where the crucible is unfolded to give a regular three-dimensional structure. This relation between the crucible and the main arrays is shown in Fig. ??.

The algorithm consists of an initialization phase, a time loop, and an output phase. In the time loop for each time step the new temperature, the pressure, and the velocity are computed and the boundary conditions are updated. The most time consuming procedure is the computation of the velocity and the pressure. Here the linear equation system resulting from the partial differential equations is solved by successive overrelaxation.

In this subroutine as well as in the other operations of the time loop mostly stencil operations are performed on the data structures, i.e. in order to compute an array element only the values of neighbouring elements are needed. When updating the boundary conditions some non-local operations are applied, such as copying plane 91 onto plane 1 and plane 2 onto plane 92, thus simulating the closed crucible.

The code is designed such that a simulation can be split into a sequence of program runs. The program generates a continuation data set at the end of a run. In addition, data can be output during the time loop to allow off-line visualization.

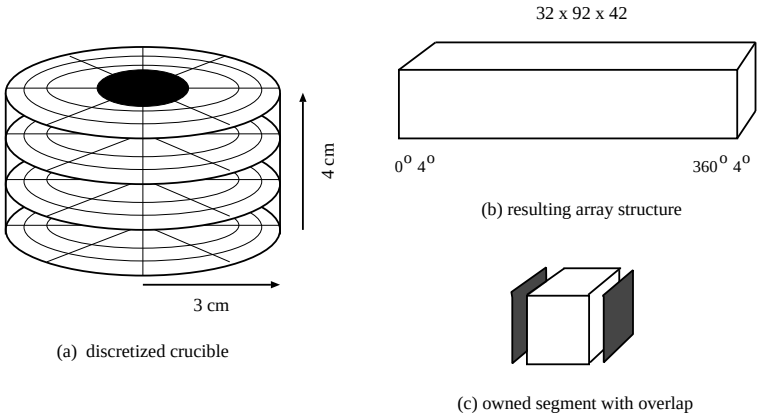


Fig. 13. Application domain

Parallelization strategy. The code is parallelized according to the data partitioning approach. The main arrays are divided into blocks that are assigned to the processors. For the application the High Performance Fortran BLOCK distribution strategy is optimal. Depending on the number of processors, a distribution of the second and the third dimension can be specified by the user.

The computation for a distributed array is spread among the processors with respect to data locality. Each processor performs the computation for its array segment. Therefore, the iterations of the loops are spread over the processors.

In this application, the distribution of arrays in blocks leads to two communication patterns: an overlap update and a remote copy. If we assume a distribution in the second dimension, the stencil operations induce access to the boundary elements of the neighbouring blocks. Therefore, the left- and rightmost plane have to be sent to the neighbouring processors prior to the computation.

The copy operation outlined in the previous section leads to communication between the rightmost and the leftmost processor, if the second dimension is distributed.

Performance results. Table ?? gives some performance results obtained on the Paragon with Paragon OSF/1 Release 1.1. The best results where obtained with a new version of the library routine *bcopy* which will be part of Release 1.2. The results demonstrate the node performance that can be expected for a memory-bound but vectorizable application, i.e. 8.3 MFLOPS, and the message-passing performance when the application becomes communication-bound with large numbers of processors. The volume-to-surface ratio is 1.7 for 72 processors, 1.5 for 112 processors, and 0.8 for 138 processors.

Table ?? shows the I/O performance for reading and writing the restart data. The I/O overhead is unimportant for long production runs but demonstrates the performance of the Parallel File System. The total amount of data is 8.8 MB.

processor configuration	execution time (s)	speedup	MFLOPS
1×1	230.0	1.0	8.3
2×1	120.7	1.9	15.7
4×1	64.8	3.5	29.3
8×1	36.3	6.4	52.3
8×2	19.2	12.0	99.0
16×2	11.8	19.5	161.0
16×4	8.7	26.4	218.4
8×9	7.3	31.5	260.3
16×7	5.7	40.4	333.3
46×3	6.5	35.4	292.3

Table 1. Execution times and speedup

The processors write one record of size 100 B and 6 records of size 150 KB for 8 processors, 35 KB for 32 processors, and 16 KB for 112 and 138 processors.

processor configuration	read pfs64	write pfs64
1×1	1.8	1.6
2×1	1.8	1.6
4×1	1.7	2.2
8×1	2.0	2.3
8×2	2.7	3.6
16×2	4.1	5.4
16×4	6.3	8.9
8×9	7.1	10.7
16×7	9.9	14.9
46×3	13.1	20.2

Table 2. Read/write restart data (PFS stripe size 64 KB)

Acknowledgement

The authors would like to thank Heinz Bast and Jörg Finger from Intel Supercomputer Systems Division for their substantial support.

References

1. Anderson, E., et al.: LAPACK User's Guide. Philadelphia, SIAM, 1992
2. Applied Parallel Research: FORGE 90, Version 8.0, User's Guide. 1992
3. Dally, W.J., Seitz, C.L.: The Torus Routing Chip. J. Distributed Computing, Vol. 1, No. 3 (1986) 187–196
4. Esser, R., Knecht, R. (eds.): Applications on KFA's Intel iPSC/860. Interner Bericht, KFA-ZAM-IB-9218, Forschungszentrum Jülich, 1992
5. Esser, R., Knecht, R.: Intel Paragon XP/S - Architecture and Software Environment. In: H.-W. Meuer (ed.): Superconputer '93. Seminar, Mannheim, 24.-26. Juni 1993, Springer-Verlag, Berlin, 1993
6. High Performance Fortran Forum: High Performance Fortran Language Specification (DRAFT), Version 0.4. Rice University, Houston TX, 1992
7. IEEE STD 1149.1-1990: IEEE Standard Test Access Port and Boundary Scan Architecture.
8. Intel Corporation: i860 Microprocessor Family Programmer's Reference Manual. Order No. 240875-002, 1992
9. Intel Supercomputer Systems Division: Software Resource Directory. Beaverton OR, October 1993
10. Intel Supercomputer Systems Division: Paragon Basic Math Library Performance Report. Order No. 312936-001, Beaverton OR, October 1993
11. Lee, K.: On the Floating Point Performance of the i860 Microprocessor. Int. J. High Speed Computing, Vol. 4, No. 4 (1992) 251–267
12. Li, K.: Shared Virtual Memory on Loosely-coupled Multiprocessors. PhD Thesis, Yale University, Technical Report YALEU-RR-492, October 1986
13. Loepere, K.: Mach 3 Kernel Principles, Open Software Foundation and Carnegie Mellon University, 1992
14. M. Mihelcic, H. Wenzl, K. Wingerath, *Flow in Czochralski Crystal Growth Melts*, Bericht des Forschungszentrums Jülich, No. 2697, ISSN 0366-0885, December 1992
15. Mlynski-Wiese, A.: Die Architektur der Prozessorfamilie i860. Interner Bericht KFA-ZAM-IB-9214, Forschungszentrum Jülich, 1992
16. Ni, L.M., McMinley, P.K.: A Survey of Wormhole Routing Techniques in Direct Networks. IEEE Computer, February 1993, 62–76
17. OSF/1 Operating System, User's Guide. Open Software Foundation, Prentice Hall, 1992

This article was processed using the LaTeX macro package with LLNCS style